Dart Programming - Operators

An expression is a special kind of statement that evaluates to a value. Every expression is composed of −

- **Operands** − Represents the data
- **Operator** − Defines how the operands will be processed to produce a value.

Consider the following expression – "2 + 3". In this expression, 2 and 3 are **operands** and the symbol "+" (plus) is the **operator**.

In this chapter, we will discuss the operators that are available in Dart.

- Arithmetic Operators
- Equality and Relational Operators
- Type test Operators
- Bitwise Operators
- Assignment Operators
- Logical Operators

Arithmetic Operators

The following table shows the arithmetic operators supported by Dart.

Sr.No	Operators & Meaning
1	**+** Add
2	**−** Subtract
3	**-expr** Unary minus, also known as negation (reverse the sign of the expression)
4	* Multiply

5	**/** Divide
6	**~/** Divide, returning an integer result
7	**%** Get the remainder of an integer division (modulo)
8	**++** Increment
9	**--** Decrement

The following example demonstrates how you can use different operators in Dart −

```
void main() {
   var num1 = 101;
   var num2 = 2;
   var res = 0;

   res = num1+num2;
   print("Addition: ${res}");

   res = num1-num2;
   print("Subtraction: ${res}");

   res = num1*num2;
   print("Multiplication: ${res}");

   res = num1/num2;
   print("Division: ${res}");

   res = num1~/num2;
   print("Division returning Integer: ${res}");

   res = num1%num2;
   print("Remainder: ${res}");

   num1++;
   print("Increment: ${num1}");
```

```
   num2--;
   print("Decrement: ${num2}");
}
```

It will produce the following output −

```
Addition:103
Subtraction: 99
Multiplication: 202
Division: 50.5
Division returning Integer: 50
Remainder: 1
Increment: 102
Decrement: 1
```

Equality and Relational Operators

Relational Operators tests or defines the kind of relationship between two entities. Relational operators return a Boolean value i.e. true/ false.

Assume the value of A is 10 and B is 20.

Operator	Description	Example
>	Greater than	(A > B) is False
<	Lesser than	(A < B) is True
>=	Greater than or equal to	(A >= B) is False
<=	Lesser than or equal to	(A <= B) is True
==	Equality	(A==B) is False
!=	Not equal	(A!=B) is True

The following example shows how you can use Equality and Relational operators in Dart −

```
void main() {
   var num1 = 5;
   var num2 = 9;
   var res = num1>num2;
   print('num1 greater than num2 ::  ' +res.toString());

   res = num1<num2;
   print('num1 lesser than  num2 ::  ' +res.toString());

   res = num1 >= num2;
   print('num1 greater than or equal to num2 ::  ' +res.toString());
```

```
   res = num1 <= num2;
   print('num1 lesser than or equal to num2  ::  ' +res.toString());

   res = num1 != num2;
   print('num1 not equal to num2 ::  ' +res.toString());

   res = num1 == num2;
   print('num1 equal to num2 ::  ' +res.toString());
}
```

It will produce the following **output** −

```
num1 greater than num2  ::  false
num1 lesser than num2  ::  true
num1 greater than or equal to num2 :: false
num1 lesser than or equal  to num2 :: true
num1 not equal to num2 :: true
num1 equal to num2  ::  false
```

Type test Operators

These operators are handy for checking types at runtime.

Operator	Meaning
is	True if the object has the specified type
is!	False if the object has the specified type

is Example

```
void main() {
   int n = 2;
   print(n is int);
}
```

It will produce the following **output** −

```
true
```

is! Example

```
void main() {
   double  n = 2.20;
   var num = n is! int;
   print(num);
}
```

It will produce the following **output** −

```
true
```

Bitwise Operators

The following table lists the bitwise operators available in Dart and their role −

Operator	Description	Example
Bitwise AND	a & b	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	a | b	Returns a one in each bit position for which the corresponding bits of either or both operands are ones.
Bitwise XOR	a ^ b	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones.
Bitwise NOT	~ a	Inverts the bits of its operand.
Left shift	a ≪ b	Shifts a in binary representation b (< 32) bits to the left, shifting in zeroes from the right.
Signpropagating right shift	a ≫ b	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off.

The following example shows how you can use the bitwise operators in Dart −

```
void main() {
   var a = 2;  // Bit presentation 10
   var b = 3;  // Bit presentation 11

   var result = (a & b);
   print("(a & b) => ${result}");
   result = (a | b);
   print("(a | b) => ${result}");
   result = (a ^ b);
   print("(a ^ b) => ${result}");

   result = (~b);
   print("(~b) => ${result}");

   result = (a < b);
   print("(a < b) => ${result}");

   result = (a > b);
   print("(a > b) => ${result}");
}
```

It will produce the following **output** −

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a > b) => false
```

Assignment Operators

The following table lists the assignment operators available in Dart.

Sr.No	Operator & Description
1	**=(Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex**:C = A + B will assign the value of A + B into C
2	**??=** Assign the value only if the variable is null
3	**+=(Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex**: C += A is equivalent to C = C + A
4	**—=(Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex**: C -= A is equivalent to C = C – A
5	***=(Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex**: C *= A is equivalent to C = C * A
6	**/=(Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand.

Note − Same logic applies to Bitwise operators, so they will become ≪=, ≫=, ≫=, ≫=, |= and ^=.

The following example shows how you can use the assignment operators in Dart −

```
void main() {
   var a = 12;
   var b = 3;

   a+=b;
   print("a+=b : ${a}");

   a = 12; b = 13;
   a-=b;
   print("a-=b : ${a}");

   a = 12; b = 13;
   a*=b;
   print("a*=b' : ${a}");

   a = 12; b = 13;
   a/=b;
   print("a/=b : ${a}");

   a = 12; b = 13;
   a%=b;
   print("a%=b : ${a}");
}
```

It will produce the following **output** −

```
a+=b : 15
a-=b : -1
a*=b' : 156
a/=b :0.9230769230769231
a%=b : 12
```

Logical Operators

Logical operators are used to combine two or more conditions. Logical operators return a Boolean value. Assume the value of variable A is 10 and B is 20.

Operator	Description	Example
&&	**And** − The operator returns true only if all the expressions specified return true	(A > 10 && B > 10) is False.
||	**OR** − The operator returns true if at least one of the expressions specified return true	(A > 10 || B > 10) is True.

!	**NOT** − The operator returns the inverse of the expression's result. For E.g.: !(7>5) returns false	!(A > 10) is True.

The following example shows how you can use Logical Operators in Dart −

```
void main() {
   var a = 10;
   var b = 12;
   var res = (a<b)&&(b>10);
   print(res);
}
```

It will produce the following **output** −

```
true
```

Let's take another example −

```
void main() {
   var a = 10;
   var b = 12;
   var res = (a>b)||(b<10);

   print(res);
   var res1 =!(a==b);
   print(res1);
}
```

It will produce the following **output** −

```
false
true
```

Short-circuit Operators (&& and ||)

The **&&** and || operators are used to combine expressions. The && operator returns true only when both the conditions return true.

Let us consider the following expression −

```
var a = 10
var result = (a<10 && a>5)
```

In the above example, **a<10 and a>5** are two expressions combined by an **&&** operator. Here, the first expression returns **false**. However, the **&&** operator requires both the expressions to return **true**. So, the operator skips the second expression.

The || operator returns **true** if one of the expressions returns **true**. For example −

```
var a = 10
var result = ( a>5 || a<10)
```

In the above snippet, two expressions **a>5** and **a<10** are combined by a **||** operator. Here, the first expression returns true. Since, the first expression returns true, the **||** operator skips the subsequent expression and returns true.

Due to this behavior of the && and || operator, they are called as short-circuit operators.

Conditional Expressions

Dart has two operators that let you evaluate expressions that might otherwise require ifelse statements −

condition ? expr1 : expr2

If condition is true, then the expression evaluates **expr1** (and returns its value); otherwise, it evaluates and returns the value of **expr2**.

The following example shows how you can use conditional expression in Dart −

```
void main() {
   var a = 10;
   var res = a > 12 ? "value greater than 10":"value lesser than or equal
             to 10"
   print(res);
}
```

It will produce the following output −

```
value lesser than or equal to 10
```

expr1 ?? expr2

If **expr1** is non-null, returns its value; otherwise, evaluates and returns the value of **expr2**

```
void main() {
   var a = null;
   var b = 12;
   var res = a ?? b;
   print(res);
}
```

It will produce the following output −

```
12
```